

Mendelova zemědělská a lesnická univerzita v Brně  
Provozně ekonomická fakulta

---

# **Interpret KOMA**

**Projekt do předmětu Teorie programovacích jazyků**

Vypracovali:

Bc. Zdeněk Loučka

Bc. Magdalena Raszková

Brno 2006

# Obsah

<b>1 Úvod</b>	<b>3</b>
<b>2 Lexikální prvky jazyka</b>	<b>4</b>
<b>3 Lexikální analýza</b>	<b>7</b>
3.1 Gramatiky pro lexikální prvky . . . . .	7
3.2 Regularizace . . . . .	9
3.3 Jazyk interpretu . . . . .	10
3.4 Stanovení významnosti lexikálních jednotek . . . . .	10
3.5 Sestavení gramatiky lexikálního analyzátoru . . . . .	10
3.6 Nedeterministický konečný automat . . . . .	12
3.7 Deterministický konečný automat . . . . .	12
3.8 Interpret lexikálního analyzátoru . . . . .	12
<b>4 Syntaktická analýza</b>	<b>13</b>
4.1 Syntaktické diagramy . . . . .	13
4.2 Syntaktická gramatika . . . . .	15
4.3 Ověření LL(1) . . . . .	17
4.4 Sestavení zásobníkového automatu . . . . .	18
4.5 Implementace syntaktické analýzy . . . . .	19
<b>5 Sémantické akce</b>	<b>19</b>
<b>6 Problémy při implementaci</b>	<b>20</b>
<b>7 Závěr</b>	<b>21</b>
<b>Přílohy</b>	<b>22</b>
<b>A Konečný automat</b>	<b>23</b>
<b>B Zásobníkový automat</b>	<b>24</b>
<b>C Testovací soubory</b>	<b>25</b>
C.1 Malá násobilka . . . . .	25
C.2 Faktoriál . . . . .	26
C.3 Pyramida . . . . .	27
C.4 Hanojské věže . . . . .	28
C.5 Kvadratická rovnice . . . . .	29
C.6 Newtonova metoda . . . . .	30

# 1 Úvod

Náš interpret jsme nazvali KOMA, což je zkratka přezdívek autorů (Kolyk + Maddy) této práce. Při návrhu jazyka KOMA jsme se částečně inspirovali syntaktickými pravidly již existujících jazyků, jako je C, Pascal či Perl, ale přináíme i některé vlastní nápady. Více o způsobu programování v jazyce KOMA lze vyčíst z následující kapitoly, kde jsou popsány základní lexikální prvky jazyka a programové struktury včetně příkladu jejich zápisu.

Interpret jazyka KOMA jsme vystavěli metodou rekurzivního sestupu. Postup konstrukce interpretu je přehledně znázorněn na následujícím schématu. Jednotlivé fáze jsou pak podrobněji popsány v následujících kapitolách dokumentace.

## 2 Lexikální prvky jazyka

Náš jazyk je netypový a není case-sensitive. V následujícím přehledu jsou popsány jednotlivé prvky jazyka společně s ukázkou jejich zápisu.

### Čísla

Jazyk podporuje celá a desetinná čísla v desítkové soustavě, čísla ve vědeckém tvaru a jiné číselné soustavy ne desítkovou nepodporujeme. Oddělovačem desetinné části čísla je tečka. Číslo může začínat symbolem +, – nebo rovnou první číslicí.

`-12.34`

### Proměnné

V zápisu identifikátorů proměnných lze použít malá i velká písmena anglické abecedy, číslice a znak podtržítka v libovolné kombinaci, přičemž identifikátor proměnné musí mít alespoň jeden znak. Proměnná navíc začínají a končí znakem \$, což slouží k lepší orientaci v rámci zdrojového kódu. Jazyk KOMA není typovým jazykem a proměnné není nutno deklarovat.

`$prom_123$`

### Klíčová slova

Klíčová slova se mohou podobně jako identifikátory proměnných skládat z malých a velkých písmen anglické abecedy, číslic a znaku podtržítka v libovolné kombinaci. Na rozdíl od proměnných však musí vždy začínat písmenem.

`NazevFunkce`

### Operátory

Podporujeme standardní aritmetické operátory, pro zbytek po celočíselném dělení slouží operátor %, pro umocnění operátor ^. Dále podporujeme logické operátory, logický AND se zapisuje pomocí &&, logický OR se zapisuje pomocí || a logické NOT pomocí !. Podporujeme i standardních relačních operátorů, rovnost se zapisuje pomocí =, nerovnost pomocí <>. Pro změnu priority lze použít kulaté závorky. Operátory zřetězení nepodporujeme.

Priorita vyhodnocování jednotlivých operátorů vyplývá se syntaktických diagramů (viz str. 13).

`$prom$ * 3 % 2 <> 1`

Ve výrazech, v nichž je druhým operandem číslo (bez znaménka) a operátorem  $+$  nebo  $-$ , je nutné tento operátor oddělit od operandů znakem mezera. Tento způsob zápisu je vyžadován z důvodu potřeby správného rozlišení arity operátorů. Nejprůhlednější je oddělovat binární operátory od operandů pomocí mezery ve všech případech (viz příklad).

```
3 + 1 + -4 - $prom$ - 2
```

### Řetězce

Řetězce se uzavírají do uvozovek `"` a mohou obsahovat libovolné znaky z abecedy interpretu. Jsou podporovány i řetězce jednoznakové či prázdné. V případě, že je v řetězci potřeba zapsat znak uvozovek nebo zpětné lomítko, je zde možnost použít zpětné lomítko, které zajistí, že znak bezprostředně po něm následující bude chápán jako obyčejná součást řetězce. Výjimkou je pouze sekvence znaku `\n`, která zajistí přechod na nový řádek.

```
"ahoj!"
```

### Oddělovače

Mezi oddělovače patří znak mezery, tabulátoru a enter.

### Komentáře

Podporujeme komentáře jednořádkové i víceřádkové. Řádkový komentář začíná kombinací znaků `[>` a je ukončen koncem řádku. Komentář víceřádkový začíná znaky `[[` a končí znaky `]]`. Jednořádkový komentář může obsahovat libovolné znaky z abecedy interpretu kromě znaku enter (tím se ukončuje), víceřádkový komentář může rovněž obsahovat libovolné znaky kromě dvojice znaků `]]`, kterými se víceřádkový komentář ukončuje.

```
[> Toto je jednořádkový komentář  
[[ Toto je víceřádkový komentář ]]
```

### Program a příkazy

Celý programový blok je uzavřen v mezi znaky `{` a `}`. Podporujeme příkaz přiřazení, cyklu, větvení, definice funkce, volání funkce, příkaz vstupu a výstupu (viz Standardní funkce). Prázdný příkaz nepodporujeme.

Příkazy pro volání funkce, přiřazení, vstup a výstup jsou ukončeny středníkem. Ostatní příkazy, vzhledem k tomu, že jejich součástí je blok, středníkem ukončovány nejsou.

Syntaxe příkazů je nejlépe patrná ze syntaktických diagramů (viz str. 13).

```
$prom$ := 123;
```

## Funkce

Funkce se definují pomocí klíčového slova **function** společně s názvem funkce a seznamem parametrů. Název funkce je ve své podstatě klíčovým slovem, proto pro něj platí pravidla pro zápis klíčových slov. Funkce může mít více parametrů, které se pak oddělují čárkou. Za touto hlavičkou následuje blok příkazů. Definice funkce musí být uvedena před prvním voláním, ale nemusí to být nutně úplně na začátku programu.

Návratová hodnota funkce se předává pomocí proměnné **\$RETURN\$**, do které se hodnota přiřadí.

Funkce či procedura se volá pomocí názvu. Procedury jsou realizovány jako funkce bez návratové hodnoty.

```
function nasob($x$, $y$) { $RETURN$ := $x$ * $y$; }  
nasob($cislo1$, $cislo2$);
```

## Standardní funkce

Součástí jazyka je několik předdefinovaných funkcí. Funkce pro vstup je uvozena klíčovým slovem **read** **read**. Jejím parametrem je identifikátor proměnné, do níž se načítá ze standardního vstupu. Funkce **write** slouží pro výpis. Jejím parametrem je logický výraz (resp. také proměnná, číslo, řetězec či návratová hodnota při volání funkce). Oba příkazy jsou ukončeny středníkem.

```
read($cislo$);  
write("Zadali jste: "); write($cislo$);
```

### 3 Lexikální analýza

Lexikální analýza byla prvním krokem při výstavbě interpretu. Navrhli jsme gramatiky pro jednotlivé lexikální prvky jazyka (tokeny). Následně jsem stanovili významnost lexikálních jednotek a sestavili gramatiky pro nevýznamové a významové tokeny. Po sloučení obou gramatik pak vznikla hutná podoba lexikálního analyzátoru, u níž jsme provedli regularizaci.

#### 3.1 Gramatiky pro lexikální prvky

##### Gramatika jazyka čísel (D)

$$G_D = (N_D, \Sigma_D, P_D, S_D)$$

$$N_D = \{S_D, D_1, D_2\}$$

$$\Sigma_D = \{c, +, -, .\} \quad c \sim 0..9$$

$$P_D :$$

$$S_D \rightarrow +D_1 \mid -D_1 \mid D_1$$

$$D_1 \rightarrow cD_1 \mid c \mid c.D_2$$

$$D_2 \rightarrow cD_2 \mid c$$

##### Gramatika jazyka operátorů (O)

$$G_O = (N_O, \Sigma_O, P_O, S_O)$$

$$N_O = \{S_O\}$$

$$\Sigma_O = \{+, -, *, /, \&, |, !, (, ), <, >, =, ^, \%\}$$

$$P_O :$$

$$S_O \rightarrow + \mid - \mid * \mid / \mid \&\& \mid || \mid ! \mid ( \mid ) \mid < \mid > \mid <= \mid >= \mid = \mid <> \mid ^ \mid \%$$

##### Gramatika jazyka přiřazení (A)

$$G_A = (N_A, \Sigma_A, P_A, S_A)$$

$$N_A = \{S_A\}$$

$$\Sigma_A = \{:, =\}$$

$$P_A :$$

$$S_A \rightarrow :=$$

##### Gramatika jazyka řetězců (S)

$$G_S = (N_S, \Sigma_S, P_S, S_S)$$

$$N_S = \{S_S, S_1, S_2\}$$

$$\Sigma_S = \{", \backslash, \Delta\}$$

$P_S :$

$$S_S \rightarrow "S_1$$

$$S_1 \rightarrow \Delta S_1 \mid " \mid \backslash S_2$$

$$S_2 \rightarrow \Delta S_1 \mid "S_1 \mid \backslash S_1$$

$$\Delta = \Sigma - \{", \backslash\}$$

### Gramatika jazyka proměnných (V)

$$G_V = (N_V, \Sigma_V, P_V, S_V)$$

$$N_V = \{S_V, V_1, V_2\}$$

$$\Sigma_V = \{p, c, -, \$\} \quad p \sim A..Z, a..z \quad c \sim 0..9$$

$P_V :$

$$S_V \rightarrow \$V_1$$

$$V_1 \rightarrow pV_2 \mid cV_2 \mid \_V_2$$

$$V_2 \rightarrow pV_2 \mid cV_2 \mid \_V_2 \mid \$$$

### Gramatika klíčových slov (R)

$$G_R = (N_R, \Sigma_R, P_R, S_R)$$

$$N_R = \{S_R, R_1\}$$

$$\Sigma_R = \{p, c, \_ \}$$

$P_R :$

$$S_R \rightarrow pR_1 \mid p$$

$$R_1 \rightarrow pR_1 \mid cR_1 \mid \_R_1 \mid p \mid c$$

### Gramatika jazyka příkazových bloků (B)

$$G_B = (N_B, \Sigma_B, P_B, S_B)$$

$$N_B = \{S_B\}$$

$$\Sigma_B = \{\{, \}\}$$

$P_B :$

$$S_B \rightarrow \{ \mid \}$$

### Gramatika ukončovacího znaku příkazu (T)

$$G_T = (N_T, \Sigma_T, P_T, S_T)$$

$$N_T = \{S_T\}$$



$$\Sigma_T = \{ ; \}$$

$$P_T : \\ S_T \rightarrow ;$$

### Gramatika oddělovače parametrů funkcí (P)

$$G_P = (N_P, \Sigma_P, P_P, S_P)$$

$$N_P = \{ S_P \}$$

$$\Sigma_P = \{ , \}$$

$$P_P : \\ S_P \rightarrow ,$$

### Gramatika jazyka bílých znaků (W)

$$G_W = (N_W, \Sigma_W, P_W, S_W)$$

$$N_W = \{ S_W \}$$

$$\Sigma_W = \{ \_, TAB, \leftarrow \}$$

$$P_W : \\ S_W \rightarrow \_ \mid TAB \mid \leftarrow$$

### Gramatika jazyka komentářů (C)

$$G_C = (N_C, \Sigma_C, P_C, S_C)$$

$$N_C = \{ S_C, C_1, C_2, C_3 \}$$

$$\Sigma_C = \{ [, > \}$$

$$P_C : \\ S_C \rightarrow [ > C_1 \mid [[ C_2 \\ C_1 \rightarrow \blacksquare C_1 \mid \leftarrow \\ C_2 \rightarrow \square C_2 \mid ] C_3 \\ C_3 \rightarrow \square C_2 \mid ]$$

$$\blacksquare = \Sigma - \{ \leftarrow \}$$

$$\square = \Sigma - \{ ] \}$$

## 3.2 Regularizace

Regularizaci lineárních gramatik pro jednotlivé lexikální prvky jazyka jsme provedli a« po jejich sloučení v celkovou gramatiku lexikálního analyzátoru.

### 3.3 Jazyk interpretu

$$\Sigma = \{A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, \\ a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \\ +, -, ., *, \&, /, \backslash, |, !, (, ), <, >, =, :, ", \{, \}, \neg, TAB, \leftarrow, ;, [, ], \neg, \$, ^, \%, \#, ,, , ?, @, \sim, ', '\}$$

Pro přehlednost jsme si zavedli:

$$p \sim A..Z, a..z$$

$$c \sim 0..9$$

### 3.4 Stanovení významnosti lexikálních jednotek

#### Nevýznamové jednotky ( $S_{od}$ )

Do nevýznamových jednotek patří bílé znaky a komentáře. Gramatika nevýznamových jednotek (oddělovačů) má následující podobu:

$$\begin{aligned} S_{od} &\rightarrow [ > C_1 \mid [[C_2 \mid \neg S_{od} \mid TABS_{od} \mid \leftarrow S_{od} \mid \epsilon \\ C_1 &\rightarrow \blacksquare C_1 \mid \leftarrow S_{od} \\ C_2 &\rightarrow \square C_2 \mid ]C_3 \\ C_3 &\rightarrow \square C_2 \mid ]S_{od} \end{aligned}$$

$$\blacksquare = \Sigma - \{\leftarrow\}$$

$$\square = \Sigma - \{ ] \}$$

#### Významové jednotky ( $S_{vt}$ )

Do významových jednotek řadíme čísla, operátory, operátor přiřazení, řetězce, identifikátory proměnných, klíčová slova, příkazový blok, ukončovací znak příkazu a oddělovač parametrů funkcí. Pro gramatiku významových jednotek platí:

$$\begin{aligned} L_{vt} &= D \cup O \cup A \cup S \cup V \cup R \cup B \cup T \\ S_{vt} &\rightarrow S_D \mid S_O \mid S_A \mid S_S \mid S_V \mid S_R \mid S_B \mid S_T \end{aligned}$$

$$\begin{aligned} S_{vt} &\rightarrow +D_1 \mid -D_1 \mid D_1 \mid + \mid - \mid * \mid / \mid \&\& \mid || \mid ! \mid ( \mid ) \mid < \mid > \mid <= \mid >= \\ &\mid = \mid <> \mid ^ \mid \% \mid := \mid "S_1 \mid \$V_1 \mid pR_1 \mid p \mid \{ \mid \} \mid ; \mid , \end{aligned}$$

### 3.5 Sestavení gramatiky lexikálního analyzátoru

#### Lineární lexikální analyzátor

Platí:  $lex \sim L_{od} * L_{vt}$

$$\begin{aligned}
S &\rightarrow [> C_1 \mid [[C_2 \mid \_S \mid TABS \mid \leftrightarrow S \mid S_{vt} \mid \epsilon \\
D_1 &\rightarrow cD_1 \mid c \mid c.D_2 \\
D_2 &\rightarrow cD_2 \mid c \\
S_1 &\rightarrow \triangle S_1 \mid " \mid \backslash S_2 \\
S_2 &\rightarrow \triangle S_1 \mid " S_1 \mid \backslash S_1 \\
V_1 &\rightarrow pV_2 \mid cV_2 \mid \_V_2 \\
V_2 &\rightarrow pV_2 \mid cV_2 \mid \_V_2 \mid \$ \\
R_1 &\rightarrow pR_1 \mid cR_1 \mid \_R_1 \mid p \mid c \\
C_1 &\rightarrow \blacksquare C_1 \mid \leftrightarrow S \\
C_2 &\rightarrow \square C_2 \mid ]C_3 \\
C_3 &\rightarrow \square C_2 \mid ]S \\
S_{vt} &\rightarrow +D_1 \mid -D_1 \mid D_1 \mid + \mid - \mid * \mid / \mid \&\& \mid || \mid ! \mid ( \mid ) \mid < \mid > \mid <= \mid >= \\
&\mid = \mid <> \mid ^ \mid \% \mid := \mid " S_1 \mid \$V_1 \mid pR_1 \mid p \mid \{ \mid \} \mid ; \mid ,
\end{aligned}$$

$$\triangle = \Sigma - \{ ", \backslash \}$$

$$\blacksquare = \Sigma - \{ \leftrightarrow \}$$

$$\square = \Sigma - \{ ] \}$$

### Regularizovaný lexikální analyzátor

Po provedení regularizace získáváme následující regulární gramatiku lexikálního analyzátoru:

$$\begin{aligned}
S &\rightarrow [C_4 \mid \_S \mid TABS \mid \leftrightarrow S \mid + D_1 \mid - D_1 \mid cD_1 \mid c \mid cD_3 \mid + \mid - \mid * \\
&\mid / \mid \&O_1 \mid |O_2| \mid ! \mid ( \mid ) \mid < \mid > \mid < O_3 \mid > O_3 \mid = \mid < O_4 \mid ^ \mid \% \mid : \\
&O_3 \mid " S_1 \mid \$V_1 \mid pR_1 \mid p \mid \{ \mid \} \mid ; \mid , \mid \epsilon \\
D_1 &\rightarrow cD_1 \mid c \mid cD_3 \\
D_2 &\rightarrow cD_2 \mid c \\
S_1 &\rightarrow \bullet S_1 \mid " \mid \backslash S_2 \mid \leftrightarrow S_1 \mid ]S_1 \\
S_2 &\rightarrow \bullet S_1 \mid " S_1 \mid \backslash S_1 \mid \leftrightarrow S_1 \mid ]S_1 \\
V_1 &\rightarrow pV_2 \mid cV_2 \mid \_V_2 \\
V_2 &\rightarrow pV_2 \mid cV_2 \mid \_V_2 \mid \$ \\
R_1 &\rightarrow pR_1 \mid cR_1 \mid \_R_1 \mid p \mid c \\
C_1 &\rightarrow \bullet C_1 \mid " C_1 \mid \backslash C_1 \mid ]C_1 \mid \leftrightarrow S \\
C_2 &\rightarrow \bullet C_2 \mid " C_2 \mid \backslash C_2 \mid ]C_3 \mid \leftrightarrow C_2 \\
C_3 &\rightarrow \bullet C_2 \mid " C_2 \mid \backslash C_2 \mid ]S \mid \leftrightarrow C_2 \\
S_{vt} &\rightarrow +D_1 \mid -D_1 \mid cD_1 \mid c \mid cD_3 \mid + \mid - \mid * \mid / \mid \&O_1 \mid |O_2| \mid ! \mid ( \mid ) \mid < \mid > \\
&\mid < O_3 \mid > O_3 \mid = \mid < O_4 \mid ^ \mid \% \mid : O_3 \mid " S_1 \mid \$V_1 \mid pR_1 \mid p \mid \{ \mid \} \mid ; \mid , \\
C_4 &\rightarrow > C_1 \mid [C_2 \\
D_3 &\rightarrow .D_2 \\
O_1 &\rightarrow \&
\end{aligned}$$

$$\begin{aligned} O_2 &\rightarrow | \\ O_3 &\rightarrow = \\ O_4 &\rightarrow > \end{aligned}$$

$$\bullet = \Sigma - \{', \backslash, \leftrightarrow, ]\}$$

### 3.6 Nedeterministický konečný automat

Posledním krokem lexikální analýzy bylo sestavení nedeterministického konečného automatu a jeho determinizace.

*Nedeterministický konečný automat je součástí přílohy A.*

### 3.7 Deterministický konečný automat

Deterministický konečný automat byl pak východiskem pro naprogramování vlastního lexikálního analyzátoru.

*Deterministický konečný automat je součástí přílohy A.*

### 3.8 Interpret lexikálního analyzátoru

Interpret lexikálního analyzátoru jsme sestavili na základě deterministického konečného automatu.

## 4 Syntaktická analýza

Sestavení syntaktického analyzátoru předchází sestavení následujících syntaktických diagramů:

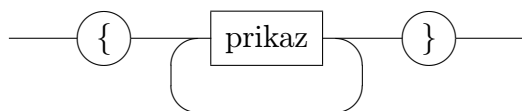
1. KOMA - celý jazyk, obsahuje jen blok;
2. blok;
3. příkaz;
4. cyklus;
5. větvení;
6. definice funkce;
7. volání funkce;
8. přiřazení;
9. vstup;
10. výstup;
11. podmínka;
12. aritmetický výraz;
13. logický výraz.

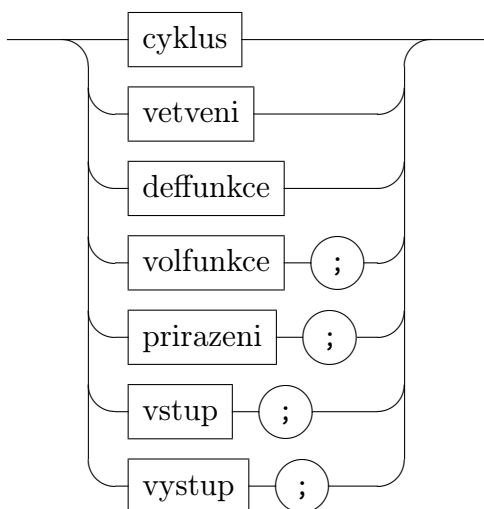
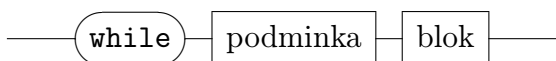
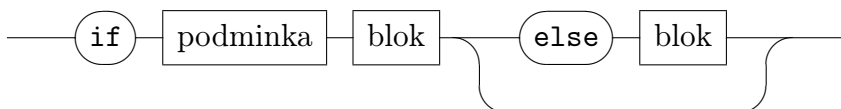
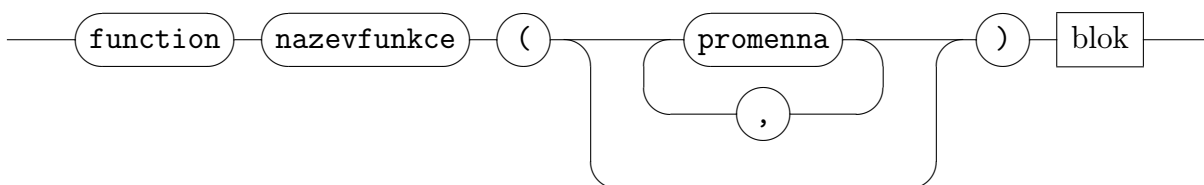
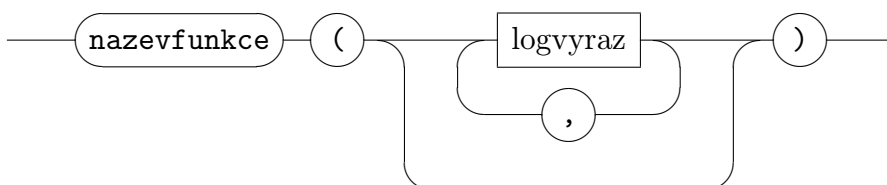
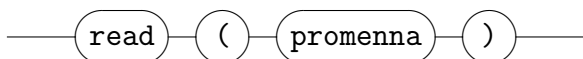
### 4.1 Syntaktické diagramy

*KOMA*

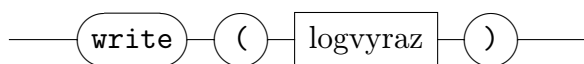


*blok*

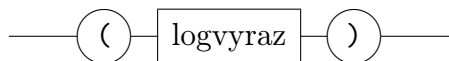


*prikaz**cyklus**vetveni**deffunkce**volfunkce**vstup*

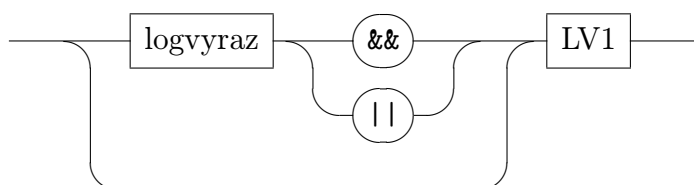
*vystup*



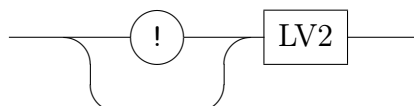
*podminka*



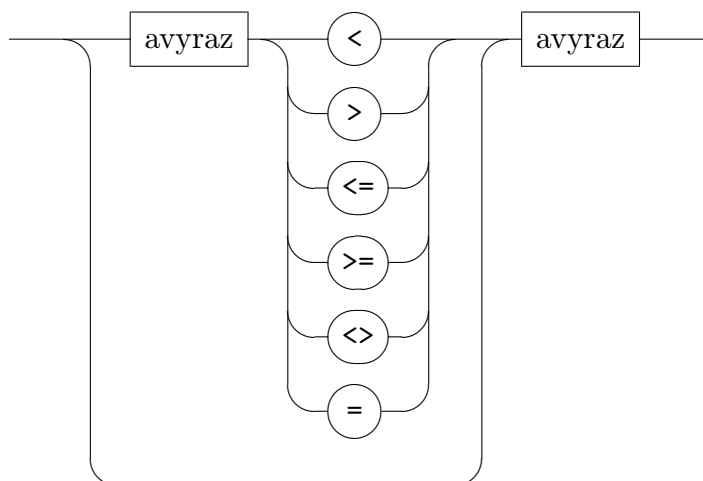
*logvyraz*



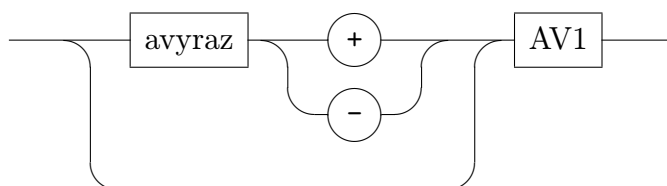
*LV1*

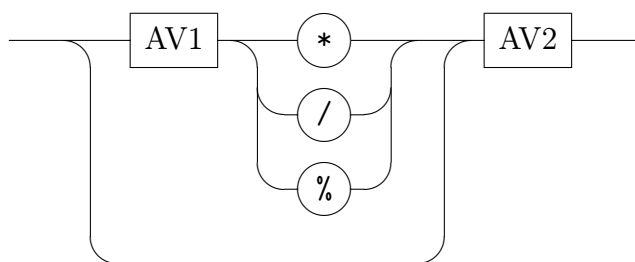
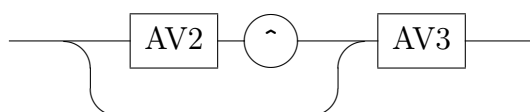
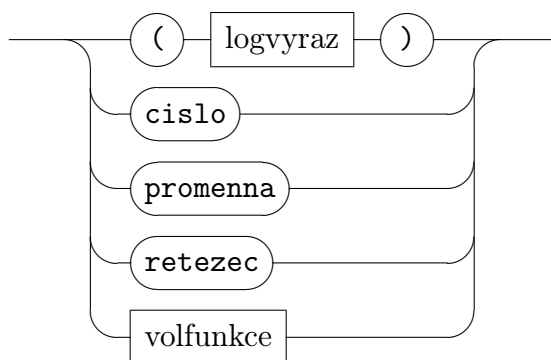


*LV2*



*avyraz*



*AV1**AV2**AV3*



## 4.2 Syntaktická gramatika

Výe uvedené syntaktické diagramy přepíeme do syntaktické gramatiky.

Vysvětlení notace:

- terminály jsou podtržené,
- neterminály jsou velkými písmeny,
- neterminály, které přibýly a« při tvorbě gramatiky a nejsou v syntaktických diagramech, jsou zvýrazněny tučně.

$KOMA \rightarrow BLOK$

$BLOK \rightarrow \{ PRIKAZ \textbf{P1}$

$PR1 \rightarrow PRIKAZ \textbf{P1} \mid \}$

$PRIKAZ \rightarrow CYKLUS \mid VETVENI \mid DEFFUNKCE \mid VOLFUNKCE; \mid$

$PRIRAZENI; \mid VSTUP; \mid VYSTUP;$

$CYKLUS \rightarrow \underline{while} \textit{PODMINKA} BLOK$

$VETVENI \rightarrow \underline{if} \textit{PODMINKA} BLOK \textbf{V1}$

$V1 \rightarrow \underline{else} BLOK \mid \epsilon$

$DEFFUNKCE \rightarrow \underline{function} \underline{nazev funkce} ( \textbf{F1} ) BLOK$

$F1 \rightarrow \underline{promenna} \textbf{P1} \mid \epsilon$

$P1 \rightarrow , \underline{promenna} \textbf{P1} \mid \epsilon$

$VOLFUNKCE \rightarrow \underline{nazev funkce} ( \textbf{F2} )$

$F2 \rightarrow LOGVYRAZ \textbf{P2} \mid \epsilon$

$P2 \rightarrow , LOGVYRAZ \textbf{P2} \mid \epsilon$

$VSTUP \rightarrow \underline{read} ( \underline{promenna} )$

$VYSTUP \rightarrow \underline{write} ( LOGVYRAZ )$

$PRIRAZENI \rightarrow \underline{promenna} := LOGVYRAZ$

$PODMINKA \rightarrow ( LOGVYRAZ )$

$LOGVYRAZ \rightarrow LOGVYRAZ \&\& LV1 \mid LOGVYRAZ \parallel LV1 \mid LV1$

$LV1 \rightarrow ! LV2 \mid LV2$

$LV2 \rightarrow AVYRAZ \underline{ro} AVYRAZ \mid AVYRAZ$

$AVYRAZ \rightarrow AVYRAZ \pm AV1 \mid AVYRAZ \pm AV1 \mid AV1$

$AV1 \rightarrow AV1 * AV2 \mid AV1 / AV2 \mid AV1 \% AV2 \mid AV2$

$AV2 \rightarrow AV2 \wedge AV3 \mid AV3$

$AV3 \rightarrow ( LOGVYRAZ ) \mid \underline{cislo} \mid \underline{promenna} \mid \underline{retezec} \mid VOLFUNKCE$

Gramatiku je potřeba dále upravit tak, aby neobsahovala bezprostřední levou rekurzi a bylo možné určit množiny First a Follow.

1.  $KOMA \rightarrow BLOK$
2.  $BLOK \rightarrow \{ \text{PRIKAZ } \mathbf{PR1}$
3.  $\mathbf{PR1} \rightarrow \text{PRIKAZ } \mathbf{PR1} \mid \}$
4.  $\text{PRIKAZ} \rightarrow \text{CYKLUS} \mid \text{VETVENI} \mid \text{DEFFUNKCE} \mid \text{VOLFUNKCE}; \mid$   
 $\text{PRIRAZENI}; \mid \text{VSTUP}; \mid \text{VYSTUP};$
5.  $\text{CYKLUS} \rightarrow \text{while } \text{PODMINKA } BLOK$
6.  $\text{VETVENI} \rightarrow \text{if } \text{PODMINKA } BLOK \mathbf{V1}$
7.  $\mathbf{V1} \rightarrow \text{else } BLOK \mid \epsilon$
8.  $\text{DEFFUNKCE} \rightarrow \text{function } \underline{\text{nazev funkce}} ( \mathbf{F1} ) BLOK$
9.  $\mathbf{F1} \rightarrow \underline{\text{promenna}} \mathbf{P1} \mid \epsilon$
10.  $\mathbf{P1} \rightarrow \text{, } \underline{\text{promenna}} \mathbf{P1} \mid \epsilon$
11.  $\text{VOLFUNKCE} \rightarrow \underline{\text{nazev funkce}} ( \mathbf{F2} )$
12.  $\mathbf{F2} \rightarrow \text{LOGVYRAZ } \mathbf{P2} \mid \epsilon$
13.  $\mathbf{P2} \rightarrow \text{, } \text{LOGVYRAZ } \mathbf{P2} \mid \epsilon$
14.  $\text{VSTUP} \rightarrow \text{read} ( \underline{\text{promenna}} )$
15.  $\text{VYSTUP} \rightarrow \text{write} ( \text{LOGVYRAZ} )$
16.  $\text{PRIRAZENI} \rightarrow \underline{\text{promenna}} := \text{LOGVYRAZ}$
17.  $\text{PODMINKA} \rightarrow ( \text{LOGVYRAZ} )$
18.  $\text{LOGVYRAZ} \rightarrow \text{LV1 } \overline{\text{LOGVYRAZ}}$
19.  $\overline{\text{LOGVYRAZ}} \rightarrow \&\& \text{LV1 } \overline{\text{LOGVYRAZ}} \mid \mid \text{LV1 } \overline{\text{LOGVYRAZ}} \mid \epsilon$
20.  $\text{LV1} \rightarrow ! \text{LV2} \mid \text{LV2}$
21.  $\text{LV2} \rightarrow \text{AVYRAZ } \overline{\text{LV2}}$
22.  $\overline{\text{LV2}} \rightarrow \text{ro } \text{AVYRAZ } \overline{\text{LV2}} \mid \epsilon$
23.  $\text{AVYRAZ} \rightarrow \text{AV1 } \overline{\text{AVYRAZ}}$
24.  $\overline{\text{AVYRAZ}} \rightarrow \pm \text{AV1 } \overline{\text{AVYRAZ}} \mid \pm \text{AV1 } \overline{\text{AVYRAZ}} \mid \epsilon$
25.  $\text{AV1} \rightarrow \text{AV2 } \overline{\text{AV1}}$
26.  $\overline{\text{AV1}} \rightarrow * \text{AV2 } \overline{\text{AV1}} \mid / \text{AV2 } \overline{\text{AV1}} \mid \% \text{AV2 } \overline{\text{AV1}} \mid \epsilon$
27.  $\text{AV2} \rightarrow \text{AV3 } \overline{\text{AV2}}$
28.  $\overline{\text{AV2}} \rightarrow \wedge \text{AV3 } \overline{\text{AV2}} \mid \epsilon$
29.  $\text{AV3} \rightarrow ( \text{LOGVYRAZ} ) \mid \text{cislo} \mid \underline{\text{promenna}} \mid \underline{\text{retezec}} \mid \text{VOLFUNKCE}$

### 4.3 Ověření LL(1)

Po úpravě gramatiky vypočítáme množiny First a Follow a ověříme, zda je jazyk typu LL(1).

1.  $FF(KOMA \rightarrow BLOK) = FI(BLOK) = \{ \{ \}$
2.  $FF(BLOK \rightarrow \{ PRIKAZ PR1) = FI(BLOK \rightarrow \{ PRIKAZ PR1) = \{ \{ \}$
3.  $FF(PR1 \rightarrow PRIKAZ PR1) = FI(PRIKAZ) =$   
 $\{ \underline{while}, \underline{if}, \underline{function}, \underline{nazev funkce}, \underline{promenna}, \underline{read}, \underline{write} \}$   
 $FF(PR1 \rightarrow \}) = \{ \} \}$
4.  $FF(PRIKAZ \rightarrow CYKLUS | VETVENI | DEFFUNKCE | VOLFUNKCE; |$   
 $PRIRAZENI; | VSTUP; | VYSTUP; ) =$   
 $\{ \underline{while}, \underline{if}, \underline{function}, \underline{nazev funkce}, \underline{promenna}, \underline{read}, \underline{write} \}$
5.  $FF(CYKLUS \rightarrow \underline{while} PODMINKA BLOK) = \{ \underline{while} \}$
6.  $FF(VETVENI \rightarrow \underline{if} PODMINKA BLOK V1) = \{ \underline{if} \}$
7.  $FF(V1 \rightarrow \underline{else} BLOK) = \{ \underline{else} \}$   
 $FF(V1 \rightarrow \epsilon) = FO(V1) = FO(VETVENI) = FO(PRIKAZ) = FI(PR1)$   
 $= \{ \underline{while}, \underline{if}, \underline{function}, \underline{nazev funkce}, \underline{promenna}, \underline{read}, \underline{write}, \} \}$
8.  $FF(DEFFUNKCE \rightarrow \underline{function} \underline{nazev funkce} ( F1 ) BLOK) = \{ \underline{function} \}$
9.  $FF(F1 \rightarrow \underline{promenna} P1) = \{ \underline{promenna} \}$   
 $FF(F1 \rightarrow \epsilon) = FO(F1) = \{ \}$
10.  $FF(P1 \rightarrow , \underline{promenna} P1) = \{ , \}$   
 $FF(P1 \rightarrow \epsilon) = FO(P1) = FO(F1) = \{ \}$
11.  $FF(VOLFUNKCE \rightarrow \underline{nazev funkce} ( F2 )) = \{ \underline{nazev funkce} \}$
12.  $FF(F2 \rightarrow LOGVYRAZ P2) = FI(LOGVYRAZ) = FI(LV1)$   
 $= \{ \{, (, \underline{cislo}, \underline{promenna}, \underline{retezec}, \underline{nazev funkce} \}$   
 $FF(F2 \rightarrow \epsilon) = FO(F2) = \{ \}$
13.  $FF(P2 \rightarrow , LOGVYRAZ P2) = \{ , \}$   
 $FF(P2 \rightarrow \epsilon) = FO(P2) = FO(F2) = \{ \}$
14.  $FF(VSTUP \rightarrow \underline{read} ( \underline{promenna} )) = \{ \underline{read} \}$
15.  $FF(VYSTUP \rightarrow \underline{write} ( LOGVYRAZ )) = \{ \underline{write} \}$
16.  $FF(PRIRAZENI \rightarrow \underline{promenna} := LOGVYRAZ) = \{ \underline{promenna} \}$
17.  $FF(PODMINKA \rightarrow ( LOGVYRAZ )) = \{ ( \}$
18.  $FF(LOGVYRAZ \rightarrow LV1 \overline{LOGVYRAZ}) = FI(LV1) = \{ \{ \} \cup FI(LV2) =$   
 $\{ \{ \} \cup FI(AVYRAZ) = \{ \{ \} \cup FI(AV1) = \{ \{ \} \cup FI(AV2) = \{ \{ \} \cup FI(AV3) =$   
 $\{ \{, (, \underline{cislo}, \underline{promenna}, \underline{retezec}, \underline{nazev funkce} \}$
19.  $FF(\overline{LOGVYRAZ} \rightarrow \&\& LV1 \overline{LOGVYRAZ}) = \{ \&\& \}$   
 $FF(\overline{LOGVYRAZ} \rightarrow \underline{\underline{\quad}} LV1 \overline{LOGVYRAZ}) = \{ \underline{\underline{\quad}} \}$

- $$FF(\overline{LOGVYRAZ} \rightarrow \epsilon) = FO(\overline{LOGVYRAZ}) = FO(LOGVYRAZ) = \{\}_{\perp} \cup FI(P2) \cup FO(PRIRAZENI) = \{\}_{\perp} \cup \{\_, \_, \_ \} \cup \{;\} = \{\_, \_, \_, ;\}$$
20.  $FF(LV1 \rightarrow ! LV2) = \{!\}$   
 $FF(LV1 \rightarrow LV2) = FI(LV2) = FI(AVYRAZ) = FI(AV1) = FI(AV2) = FI(AV3) = \{(\_, \underline{cislo}, \underline{promenna}, \underline{retezec}, \underline{nazevfunkce})\}$
  21.  $FF(LV2 \rightarrow AVYRAZ \overline{LV2}) = FI(AVYRAZ) = FI(AV1) = FI(AV2) = FI(AV3) = \{(\_, \underline{cislo}, \underline{promenna}, \underline{retezec}, \underline{nazevfunkce})\}$
  22.  $FF(\overline{LV2} \rightarrow \underline{ro} AVYRAZ \overline{LV2}) = \{\underline{ro}\}$   
 $FF(\overline{LV2} \rightarrow \epsilon) = FO(\overline{LV2}) = FO(LV2) = FO(LV1) = FI(\overline{LOGVYRAZ}) = \{\&\&, \underline{\|}, \_, \_, \_, ;\}$
  23.  $FF(AVYRAZ \rightarrow AV1 \overline{AVYRAZ}) = FI(AV2) = FI(AV3) = \{(\_, \underline{cislo}, \underline{promenna}, \underline{retezec}, \underline{nazevfunkce})\}$
  24.  $FF(\overline{AVYRAZ} \rightarrow \pm AV1 \overline{AVYRAZ}) = \{\pm\}$   
 $FF(\overline{AVYRAZ} \rightarrow \pm AV1 \overline{AVYRAZ}) = \{\pm\}$   
 $FF(\overline{AVYRAZ} \rightarrow \epsilon) = FO(\overline{AVYRAZ}) = FO(AVYRAZ) = FI(\overline{LV2}) = \{\underline{ro}, \&\&, \underline{\|}, \_, \_, \_, ;\}$
  25.  $FF(AV1 \rightarrow AV2 \overline{AV1}) = FI(AV2) = FI(AV3) = \{(\_, \underline{cislo}, \underline{promenna}, \underline{retezec}, \underline{nazevfunkce})\}$
  26.  $FF(\overline{AV1} \rightarrow * AV2 \overline{AV1}) = \{*\}$   
 $FF(\overline{AV1} \rightarrow / AV2 \overline{AV1}) = \{/ \}$   
 $FF(\overline{AV1} \rightarrow \% AV2 \overline{AV1}) = \{\%\}$   
 $FF(\overline{AV1} \rightarrow \epsilon) = FO(\overline{AV1}) = FO(AV1) = FI(\overline{AVYRAZ}) = \{\pm, \pm, \underline{ro}, \&\&, \underline{\|}, \_, \_, \_, ;\}$
  27.  $FF(AV2 \rightarrow AV3 \overline{AV2}) = FI(AV3) = \{(\_, \underline{cislo}, \underline{promenna}, \underline{retezec}, \underline{nazevfunkce})\}$
  28.  $FF(\overline{AV2} \rightarrow \wedge AV3 \overline{AV2}) = FI(\overline{AV2}) = \{\wedge\}$   
 $FF(\overline{AV2} \rightarrow \epsilon) = FO(\overline{AV2}) = FO(AV2) = FI(\overline{AV1}) = \{*, /, \%, \pm, \pm, \underline{ro}, \&\&, \underline{\|}, \_, \_, \_, ;\}$
  29.  $FF(AV3 \rightarrow (\underline{LOGVYRAZ}) | \underline{cislo} | \underline{promenna} | \underline{retezec} | \underline{VOLFUNKCE}) = \{(\_, \underline{cislo}, \underline{promenna}, \underline{retezec}, \underline{nazevfunkce})\}$

Výpočtem mno«in First a Follow jsme zjistili, «e gramatika je typu LL(1).

#### 4.4 Sestavení zásobníkového automatu

Na základě mno«in First a Follow určených pro jednotlivá pravidla jsme sestavili rozkladovou tabulku zásobníkového automatu.

*Zásobníkový automat je součástí přílohy B.*

## 4.5 Implementace syntaktické analýzy

Syntaktický analyzátor jsme implementovali metodou rekurzivního sestupu. Metoda modeluje činnost zásobníkového automatu na základě modelu rozkladové tabulky. Zásobník automatu představuje systémový zásobník pou«itý při volání procedur. Pro ka«dý neterminál je sestavena procedura, její« tělo odpovídá jednotlivým prvkům daného řádku rozkladové tabulky.

## 5 Sémantické akce

Mezi syntaktickými procedurami jsou volány i procedury představující sémantické akce, které realizují výstup překladu.

## 6 Problémy při implementaci

Problémy jsou v«dy a vude. Nejinak tomu bylo i při tvorbě tohoto projektu.

### Nedostatky v teoretickém návrhu

A« při vlastní implementaci jsme objevili některé nedostatky v teoretickém návrhu. V případě implementace lexikálního analyzátoru jsme narazili na problém se znakem čárka jako oddělovačem parametrů funkcí. Bylo nutné pro tento oddělovač sestavit gramatiku a přidat jej do významových tokenů. Také jsme opomenuli některé méně používané ASCII symboly, které se mohou vyskytnout v řetězcích a komentářích, které bylo třeba doplnit do konečného automatu.

Při návrhu syntaktického analyzátoru jsme se potýkali s problémy při snaze o udržení gramatiky v LL(1) formě. Z tohoto důvodu jsme například přistoupili k vypuštění prázdného příkazu. Rovněž jsme a« při vlastní implementaci SA provedli několik úprav v syntaktických diagramech, jako bylo ukončení některých příkazů pomocí znaku středník či vypuštění klíčových slov `then` ve větvení a `do` v cyklu, které se ukázali jako nadbytečné.

### Problémy s operátory

Zásadním problémem byl způsob rozlišení významu znaků `+` a `-` jako unárních či binárních operátorů. Na úrovni lexikálního analyzátoru byl totiž výraz typu `3 - 2` nekorektně vrácen jako dva tokeny `3` a `-2` typu číslo, což přinášelo značné komplikace.

Způsobů řešení tohoto problému se nabízelo několik. Nakonec jsme zvolili přístup, kdy je vy«adováno, aby ve výrazech, v nichž je druhým operandem číslo (bez znaménka) a operátorem `+` nebo `-`, byl operátor oddělen od operandů znakem mezera. Tento požadavek také přispívá k větší přehlednosti zdrojového kódu programů, i když připoutíme, že uvedené řešení není zcela optimální.

### Problémy s ukazateli

Při implementaci sémantických akcí prostřednictvím dynamických struktur se občas vyskytovaly problémy s ukazateli. Ty byly větinou zapříčiněny tím, že jsme opomenuli nastavit hodnotu ukazatele na `NULL`. Odladění sémantických akcí bylo vůbec nejpracnější částí implementace interpretu.

## 7 Závěr

Cílem tohoto projektu byl teoretický návrh a implementace interpretu vlastního jazyka. Projekt jsme započali sestavením lineárních gramatik pro jednotlivé lexikální prvky jazyka, přičemž gramatiky jsme následně sloučili a provedli regularizaci. Na základě celkové regulární gramatiky jsme pak sestavili nedeterministický konečný automat a provedli jeho determinizaci. Deterministický konečný automat byl pak podkladem pro implementaci lexikálního analyzátoru. Tím jsme zakončili první část výstavby interpretu, která nepřinesla větší problémy.

Následovala syntaktická analýza, kdy jsme nejprve navrhli syntaktické diagramy, které jsme přepsali do syntaktické gramatiky. Dalším poměrně náročným krokem bylo ověření, zda je jazyk typu LL(1) pomocí výpočtu množin First a Follow. Na základě množin First a Follow jsme pak sestavili zásobníkový automat, který byl podkladem pro implementaci syntaktického analyzátoru.

Poslední a časově nejnáročnější fází bylo doplnění a odladění sémantických akcí, které realizují vlastní výstup překladu.

Při konstrukci interpretu se neustále prolínaly fáze úprav teoretického návrhu a vlastního implementačního řešení.

Výsledkem je funkční interpret jazyka KOMA, který splňuje požadavky uvedené v zadání projektu. Schopnosti interpretu jsou samozřejmě omezené, nejsou podporovány souborové operace a operace s řetězci, avak pro zpracování jednoduchých programů je plně dostačující.

Celkově lze projekt shrnout jako jednoznačně přínosný, avak velice náročný a to zvláště z programátorského hlediska.

## **Přílohy**



## **A Konečný automat**

## **B Zásobníkový automat**

## C Testovací soubory

Příloha obsahuje zdrojové kódy esti ukázkových programů, napsaných v jazyce KOMA.

### C.1 Malá násobilka

Program na základě zadaného čísla vypíše malou násobilku.

```
{
[> Mala nasobilka

    write("Zadejte cislo, pro nez chcete vypsát malou násobilku: ");
    read($cislo$);

    $pom$ := 1;

    write("-----\n");
    write("Mala nasobilka:\n");

    while ($pom$ <= 10){

        write($cislo$);
        write(" * ");
        write($pom$);
        write(" = ");
        write($cislo$ * $pom$);
        write("\n");

        $pom$ := $pom$ + 1;
    }
}
```

## C.2 Faktoriál

Program vypočítá faktoriál zadaného čísla pomocí rekurze a pomocí iterace a vypíše výsledek.

```
{
[> Faktorial pomoci rekurze
    function faktorial($x$){
        if ($x$ <= 1) {$RETURN$ := 1;}
        else{$RETURN$ := $x$ * faktorial($x$ - 1);}
    }

[> Faktorial pomoci iterace
    function ifaktorial($x$){
        $pom$ := 1;
        $RETURN$ := 1;
        while($x$ > 1){
            $pom$ := $pom$ * $x$;
            $x$ := $x$ - 1;
            $RETURN$ := $pom$;
        }
    }

[> Hlavni program
    write("Zadejte kladne cislo pro vypocet faktorialu: ");
    read($cislo$);

    if($cislo$ < 0) {
        write("Zadali jste zaporne cislo! Cislo musi byt kladne.");
    }
    else{
        $fakt$ := faktorial($cislo$);
        $ifakt$ := ifaktorial($cislo$);
        write("Faktorial cisla "); write($cislo$); write(" je\n");
        write("-- pomoci rekurze: "); write($fakt$); write("\n");
        write("-- pomoci iterace: "); write($ifakt$); write("\n");
    }
}
```

## C.3 Pyramida

Program si vyžádá od uživatele počet řádků pyramidu a znak, z něhož bude pyramida postavena. Následně vypíše symetrickou pyramidu tak, že na vrcholu je jeden znak, v druhém řádku dva znaky atd.

```
{
[> Pyramida

write("Zadejte pocet radku pyramidu: ");
read($pocet$);
write("Zadejte znak: ");
read($znak$);

$radek$ := 1;
while($radek$ <= $pocet$){

    $pom$ := $pocet$ - $radek$;
    $pocatek$ := $radek$ - 1;

    while($pom$ > 0){
        write(" ");
        $pom$ := $pom$ - 1;
    }

    write($znak$);

    if($radek$ > 1){
        while($pocatek$ > 0){
            write(" ");
            write($znak$);
            $pocatek$ := $pocatek$ - 1;
        }
    }
    write("\n");
    $radek$ := $radek$ + 1;
}
```

## C.4 Hanojské věže

Program provádí přesun věže složené z disků mezi dvěma trny pomocí rekurzivního postupu. Výstupem je seznam tahů (odkud kam byl disk přesunut).

```
{
[> Hanojske veze

function disk_presun($sundej$, $nasad$) {
    write("trn ");
    write($sundej$);
    write(" --> ");
    write("trn ");
    write($nasad$);
    write("\n");
}

function vez_presun ($vyska$, $odkud$, $pomoci$, $kam$) {
    if ($vyska$>0){
        vez_presun($vyska$ - 1, $odkud$, $kam$, $pomoci$);
        disk_presun($odkud$, $kam$);
        vez_presun($vyska$ - 1, $pomoci$, $odkud$, $kam$);
    }
}

write("Zadejte pocet disku: ");
read($disky$);

write("\nPresun disku (odkud --> kam)\n");
write("-----\n");
vez_presun($disky$,1,2,3);
}
```

## C.5 Kvadratická rovnice

Program spočítá kořeny kvadratické rovnice pro zadané koeficienty.

```
{
[> Vypocet korenu kvadraticke rovnice

write("Tento program vypocita koreny kvadraticke rovnice.");
write("Zadejte koeficient a: ");
read($a$);
write("Zadejte koeficient b: ");
read($b$);
write("Zadejte koeficient c: ");
read($c$);

$diskr$ := $b$ * $b$ - 4 * $a$ * $c$;

if($diskr$ < 0){
  writeln("Rovnice ma komplexni koreny.");
}
else{
  if($a$ = 0){
    write("Rovnice je linearni s korenem x = ");
    write(-1 * $c$ / $b$);
  }
  else{
    write("Prvni koren = ");
    write( (-1 * $b$ + ($diskr$ ^ (1/2))) / (2 * $a$) );

    write("Druhy koren = ");
    write( (-1 * $b$ - ($diskr$ ^ (1/2))) / (2 * $a$) );
  }
}
}
```

## C.6 Newtonova metoda

Program si vyžadá číslo, ze kterého má být spočtena třetí odmocnina a požadovaný počet desetinných míst přesnosti. Nejprve dojde k přepočtu přesnosti a následně je proveden vlastní výpočet třetí odmocniny. Nakonec program vypíše výsledek a počet kroků, po nichž bylo výsledku dosaženo.

```
{
[> Newtonova metoda

[> Nacteni odmocnovaneho cisla
$cislo$ := -1;
while($cislo$ <= 0){
    write("Zadejte odmocnovanou hodnotu (> 0): ");
    read($cislo$);
    if($cislo$ <= 0){
        write("Spatne zadani!");
    }
}

[> Nacteni presnosti vypoctu
$presnost$ := -1;
while($presnost$ < 0){
    write("Zadejte pocet desetinnnych mist presnosti (>= 0): ");
    read($presnost$);
    if($presnost$ < 0){
        write("Spatne zadani!");
    }
}

[> Prepocet presnosti
$eps$ := 1;
while($presnost$ <> 0){
    $eps$ := $eps$ / 10;
    $presnost$ := $presnost$ - 1;
}
```



```
[> Vlastni vypocet
$x1$ := 1;
$x$ := 0;
$krok$ := 0;
$pom$ := 1;

while($pom$ > $eps$){
    $x$ := $x1$;
    $x1$ := $x$ + ($cislo$ / ($x$ * $x$) - $x$) / 3;
    $krok$ := $krok$ + 1;
    $pom$ := $x1$ - $x$;
    if ($pom$ < 0){$pom$ := $pom$ * -1;}
}

write("Treti odmocnina z ");
write($cislo$);
write(" je ");
write($x1$);
write(".\n");

write("Vysledku bylo dosazeno po ");
write($krok$);
write(" krocich");
write(".\n");
}
```